

On the visualization of the detected communities in dynamic networks: A case study of Twitter's network

Youcef Abdelsadek^{a,*}, Kamel Chelghoum^a, Francine Herrmann^a, Imed Kacem^a, Benoît Otjacques^b

^a*Laboratoire de Conception, Optimisation et Modélisation des Systèmes
LCOMS EA 7306, Université de Lorraine, Metz, France*

^b*e-Science Research Unit, Environmental Research and Innovation
Luxembourg Institute of Science and Technology, Belvaux, Luxembourg*

Abstract

Understanding the information behind social relationships represented by a network is very challenging, especially, when the social interactions change over time inducing updates on the network topology. In this context, this paper proposes an approach for analysing dynamic social networks, more precisely for Twitter's network. Our approach relies on two complementary steps: (i) an online community identification based on a dynamic community detection algorithm called *Dyci*. The main idea of *Dyci* is to track whether a connected component of the weighted graph becomes weak over time, in order to merge it with the "dominant" neighbour community. Additionally, (ii) a community visualization is provided by our visualization tool called *NLCOMS*, which combines between two methods of dynamic network visualization. In order to assess the efficiency and the applicability of the proposed approach, we consider real-world data of the ANR-Info-RSN project, which deals with community analysis in Twitter.

Keywords: dynamic networks, community detection, community visualization, visualization tool, Twitter's networks

1. Introduction

With the popularization of social networks like Twitter, an exponential quantity of data is generated and has led to the arise of relatively new field called, social network analysis [37]. These data are increasing each day, and the existing approaches which are not considering the dynamic nature of data would suffer

*Corresponding author

Email addresses: youcef.abdelsadek@univ-lorraine.fr (Youcef Abdelsadek), kamel.chelghoum@univ-lorraine.fr (Kamel Chelghoum), francine.herrmann@univ-lorraine.fr (Francine Herrmann), imed.kacem@univ-lorraine.fr (Imed Kacem), benoit.otjacques@list.lu (Benoît Otjacques)

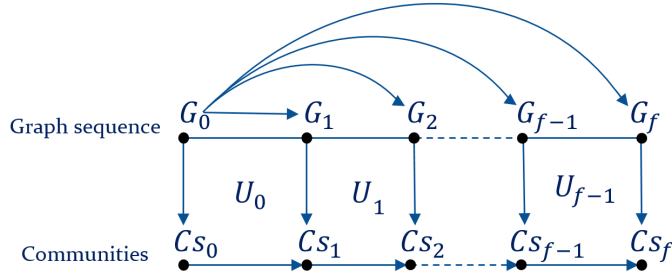


Figure 1: The graph sequence of a dynamic graph

from the scalability issue. These social relationships can be appropriately represented by networks [25, Chapter 10]. For example, we can cite social relationships like the friendship, the follow in social blogs or the information sharing in social media. Commonly, social relationships are not static but could change over time with, appearing and/or disappearing relationships for binary relationships (e.g., friendship) or increasing and/or decreasing for weighted relationships (e.g., the number of times where two persons share information). This evolving complex networks are called dynamic graphs. In this paper, we consider edge weighted dynamic graphs, where a weight is assigned to each edge of the graph to model a well-known relation in Twitter, which is the retweet. The latter occurs when a Twitter user republishes an original tweet of another Twitter user. Therefore, the edge weight corresponds to the number of times where a retweet is observed between two Twitter users. One among the objectives of this paper is to reveal the more interconnected group of nodes, known as communities, over time. In other words, the purpose of this work is to figure out the evolution of the most active groups of Twitter users.

More formally, a dynamic graph of an initial graph G_0 can be seen as a sequence of static graphs [15], denoted by $G_s = (G_0, G_1, \dots, G_f)$ with f snapshots as results of the updates $U_s = (U_0, U_1, \dots, U_{f-1})$ as illustrated in Figure 1. We denote by N_t , E_t , E_t^w , N_s and E_s respectively, the set of nodes of size v at instant t , the set of edges at instant t , the set of edge weights at instant t of G_t , the set of nodes of the whole G_s and the set of edges of the whole G_s . Furthermore, the set of updates U_t varies in terms of the impact they cause to the current network structure. As an instance, the impact of adding a new node and those of updating the weight of an existing edge differ. We point out that weights can be assigned to the nodes also, with node weight update scenario for the dynamic context, which is not considered in this paper. The following updates cases describes the repercussion on N_t , E_t and E_t^w after each update scenario.

1. Structural updates:

- (a) **Node removing:** An old node on is removed, $N_{t+1} \leftarrow N_t \setminus \{on\}$ with the related edges.

- (b) **Edge removing:** An old edge oe is removed, $E_{t+1} \leftarrow E_t \setminus \{oe\}$.
 - (c) **Node addition:** A new node nv is added, $N_{t+1} \leftarrow \{nv\} \cup N_t$ with the related edges.
 - (d) **Edge addition:** A new edge ne is added, $E_{t+1} \leftarrow \{ne\} \cup E_t$.
2. **Attributes updates:**
- (a) **Edge weight updating:** An new edge weight ne^w of an old edge weight oe^w is updated, $E_{t+1}^w \leftarrow (E_t^w \setminus \{oe^w\}) \cup \{ne^w\}$.

Furthermore, one among the important objectives in social network analysis is to reveal the semantic aspects behind the network topology. This objective can be reached by identifying the highly intra-connected groups of nodes or communities, which are in the opposite poorly inter-connected, known as community detection problem [17]. This problem is very challenging and can be found in several domains. For example, it can be met in sociology [18] where we aim to study the common characteristics of social groups and what makes groups of people together in a community. However, the community detection problem becomes trickier when the network topology changes over time. The changes that might occur to the community set $Cs_s = (Cs_0, Cs_1, \dots, Cs_f)$ can affect either the structure, the attributes (i.e., weights) or also both of them [21]. Consequently, how to analyse the evolution of the communities structure over time? To answer this question, one need to devise an algorithm, which relies on the graph features and which takes advantage from the previously identified communities by avoiding the community identification from scratch at each instant. As a concrete example, an analyst needs to understand how the information is shared in Twitter by understanding the role of each Twitter user within its community and outside of its community. To fulfil this need, one have to detect the communities of the analyst's time point of interest and to follow the community's member evolution with new members joining/leaving the studied communities. In this context, a trade-off between efficiency and response time is necessary to detect the community evolution over time.

As previously mentioned, one among the major task that we would like to carry out is to understand the community structure behaviour over time. This drives us to deal with very challenging field, which is related to the representation of relational and dynamic data using node-link diagrams as depictions, called dynamic graph drawing [14]. Recently, dynamic graph drawing and visualization gain more interest from the scientific community. They raises several new challenges additionally to challenges of the static context leading to harder tasks. For example, how to recognize with least cognitive effort the changes that occur in the graph structure? If only a small part of the graph changes, how to conserve the stability of the other parts of the graph? In order to fulfil these requirements, one have to make a trade-off between stability and efficiency. This is what we try to target in order to reach the aforementioned objectives. In the context of the previously introduced Twitter's network case. The Twitter members can retweet each others at different points of time and with different frequencies. Indeed, the analyst could want to perceive user's activity with the induced consequences on the community partition.

The remaining part of this paper is organized as follows: In Section 2, the related work of the addressed topics are presented. Section 3 introduces the *Dycki* algorithm for community identification in dynamic graphs while a genetic algorithm for the community detection problem is introduced in Section 4. Section 5 presents the case study and the conducted experiments on real-world data of the ANR Info-RSN project. In Section 6, the description of the used methods for dynamic graph visualization is presented. Finally, Section 7 concludes this paper and discuss the future improvements of this work.

2. Related work

2.1. Community detection

This section presents some related works for the dynamic community detection. There are more algorithms for the static version of this problem in the literature compared to the dynamic case, especially, for those considering weighted edges. The static community detection algorithms can be divided into two families of algorithms: the divisive family (top-down) [27] and the agglomerative family (bottom-up) [12]. The dynamic community detection problem was proved in [34] to be *NP*-complete and *APX*-hard. For the unweighed dynamic community detection, the authors of in [33] propose a matching algorithm to detect similar communities over the snapshots of the graph sequence, forming a meta-community, which is the sequence of these identical communities. Agglomerative modularity-based approach are considered in [8], [3] and [28]. The authors of [28] use a physical principle with forces, which retains a node to stay in its community against attracting forces of the other communities. Furthermore, game-theory analogy is used in [4]. In the latter, each node of the graph is considered as an agent, which maximizes its utility function. A set of predefined agent actions is set initially. The system ends when all agents choose their best community belonging (i.e., which maximizes the utility function). Relying on the colouring problem, a constant-approximation algorithm was proposed in [35]. The authors of [22] deal with changes tracking of communities in large networks. They propose an approach which uses agglomerative clustering to examine the evolution of the community structure over time by identifying stable communities after several cluster running. In [20] a model is described which tracks communities over time, those are characterised by a set of events. Regarding the weighted version of this problem, label propagation is used [38]. The idea of this algorithm is to allow a specific node to change its community label taking into account its adjacent nodes labels.

2.2. Drawing and visualization of dynamic graphs

This section reports some of the existing dynamic graphs drawing [14] and visualisation techniques [10]. Force-directed algorithms are well-known to be appropriate for static graph drawing. They produce pleasing layouts which meet the graph drawing aesthetic criteria. For example, the edge crossing minimization and the uniformity of edge length have been considered. Furthermore, they

are also appropriate for drawing dynamic graphs leading to additional aesthetic criteria [9]. One among the objectives of drawing dynamic graphs is to maintain the familiarized graph structure that the user built up over time, so-called user mental map [5]. Generally, its preservation through the successive snapshots helps the user to stay familiar with the graph structure. The mental map preservation also avoids to get the user away from its original task, which is the graph structure analysis. In [19], the authors devised a GPU-based implementation algorithm for on-line dynamic drawing. The idea of their revisited force-directed algorithm is to assign weights to graph nodes by encoding the node's movement rigidity. This means that the greater the node weight is, the less the node is flexible. The latter ensures the user mental map preservation.

After the graph is embedded, comes the visualization issue. Indeed, visualizing dynamic graphs and dynamic data in general is challenging [2]. There are two major techniques to visualize dynamic graphs. The first one relies on the physical time differing the layout of each snapshot, whereas in the second technique, an axis is used to present the successive snapshots. Several experiments have been performed in order to assess the efficiency of each technique [5][7]. In the former technique, the whole depiction area is dedicated to one snapshot. Additionally, the animation can be used in order to provide smooth snapshots transition. For example, in [16] the authors use a fade out effect for node removing. In the latter technique the depiction area is divided either in grid format or time-line, so-called small-multiple [36] where the user can follow the graph structure evolution by respecting a precedence constraint on an axis. Furthermore, these chronologically embedded snapshots can be visualized on a 2.5 dimensions with snapshot superposition [31]. In [6] the authors layout the union of two successive snapshots in one aggregated snapshot in order to highlight the difference between them relying on colors as visual variable, called difference map.

A combination of these techniques in multiple views is possible, like in Diffani [32] with difference map and animation.

3. Dynamic community detection algorithm

3.1. Notations and definitions

Let us define c_{n_i} , IW , INW , WD and WI which, respectively, represent the community of the node n_i , the intra-community weight, the inter-community weight, the weighted degree of a node and the weighted community-incidence of a node. These are presented in the following equations:

$$IW_{c_g} = \sum_{n_i \in c_g} \sum_{n_j \in c_g} \frac{e_{n_i, n_j}^w}{2} \quad (1)$$

$$INW_{c_g, c_h} = \sum_{n_i \in c_g} \sum_{n_j \in c_h} e_{n_i, n_j}^w \quad (2)$$

$$WD_{n_i} = \sum_{j=1}^v e_{n_i, n_j}^w \quad (3)$$

$$WI(node, c_g) = \frac{\sum_{n_i \in c_g} e_{node, n_i}^w}{WD_{node}} \quad (4)$$

3.2. *Dyci* algorithm

First of all, the static community detection algorithm proposed in [1] is applied on G_0 as initial community partition Cs_0 of *Dyci*. The main idea of this algorithm consists in using a collection of triangles as a starting point. Then, adjacent communities are iteratively compared in terms of weights and merged when a merging condition holds. This iterative process ends when no community merging is observed. However, for the other snapshot of the graph sequence, *Dyci* reacts depending on the update scenario as presented in the Algorithm 1.

Algorithm 1 *Dyci* algorithm

Input: G_t , Cs_t and U_t ;

Output: Cs_{t+1} .

BEGIN

if G_t is G_0 **then**

 Launch the static community detection algorithm *Tribase* [1].

else

for each $oldNode$ **in** $U_t.nodeToRemove$ **do**

$Cs_t \leftarrow NodeRemoving (oldNode, Cs_t, G_t);$

end for

for each $oldEdge$ **in** $U_t.edgeToRemove$ **do**

$Cs_t \leftarrow EdgeRemoving (oldEdge, Cs_t, G_t);$

end for

for each $newNode$ **in** $U_t.nodeToAdd$ **do**

$Cs_t \leftarrow NodeAddition (newNode, Cs_t, G_t);$

end for

for each $newEdge$ **in** $U_t.edgeToAdd$ **do**

$Cs_t \leftarrow EdgeAddition (newEdge, Cs_t, G_t);$

end for

for each $edgeWeightUpdate$ **in** $U_t.edgeWeightUpdate$ **do**

$Cs_t \leftarrow EdgeWeightUpdating (newEdgeWeight, Cs_t, G_t);$

end for

end if

Return Cs_t ;

END.

The following subsections show how *Dyci* reacts depending on the update scenario. Each update case is considered and presented in detail with the related illustrative example.

3.2.1. NodeRemoving (*oldNode*):

The main idea of the node removing case is to check whether the deletion of *oldNode* generates several connected components or reduces the $IW^{(oldNode)}$. To this end, *Dyci* tests for each resulting connected component, noted CC , whether it can form a community by it self or would be merged with an adjacent community, noted com . In other words, *Dyci* verifies whether Equation 5 holds or not. Figure 2 gives an example of the node removing update scenario.

$$INW_{com,CC} \geq IW_{CC} \quad (5)$$

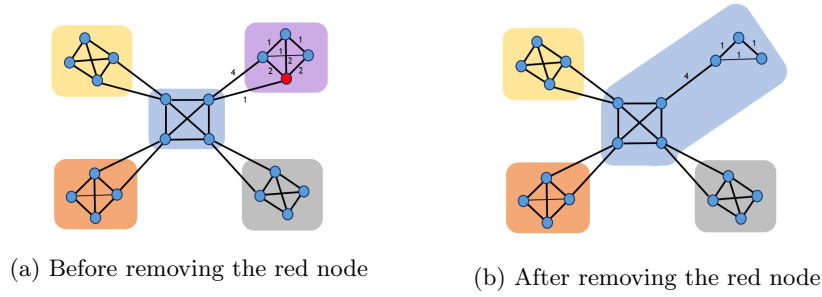


Figure 2: Node removing scenario example

3.2.2. EdgeRemoving (*oldEdge*):

Dyci handles the edge removing scenario as follows. Obviously, when an inter-community edge is removed, this reduces the inter-community weight leading to more community-like structure. However, the opposite case might lead to intra-community dividing in two connected components or a significant weight loss. To manage this second case, *Dyci* compares the weights between each resulting connected component of *oldEdge* deletion and their adjacent communities by Equation 5. The edge removing update scenario is illustrated with an example in Figure 3.

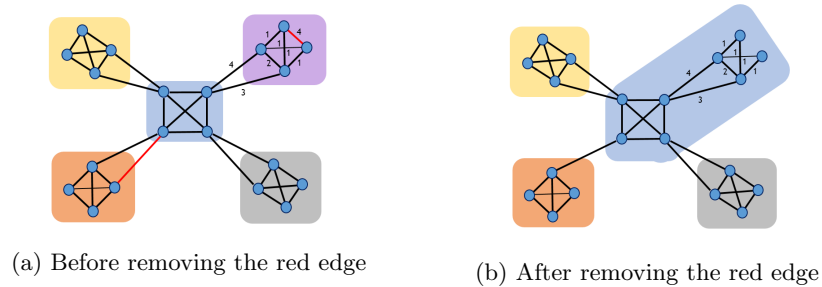


Figure 3: Edge removing scenario example

3.2.3. NodeAddition (*newNode*):

Two subcases can occur for node addition scenario. The first one, which can be viewed as trivial, is the subcase where *newNode* has no community edge incidence leading to an isolated community. In the second subcase, which is more common, *newNode* comes with many edges. For the latter, *newNode* is added to the community with the greatest $WI(newNode, c)$, where c is a community adjacent to *newNode*. Figure 4 gives an example of the node addition update scenario.

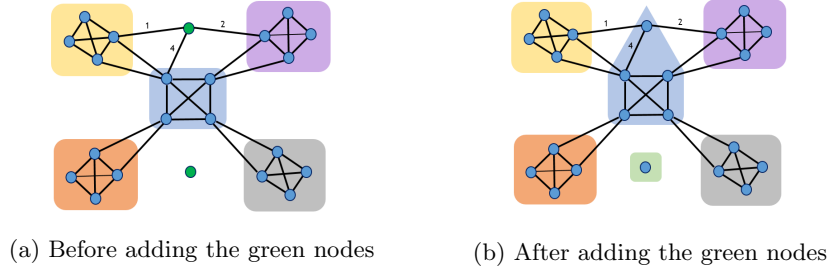


Figure 4: Node addition scenario example

3.2.4. EdgeAddition (*newEdge*):

For this case, if *newEdge* is inserted inside a community, this will not affect the community partition in terms of weights. Unlike, an inter-community edge could increase the inter-community weight, noted c_1 and c_2 , aggregating them in one community. To handle this case, *Dyci* verifies whether Equation 6 holds or not. The edge addition update scenario is illustrated with an example in Figure 5.

$$INW_{c_1, c_2} \geq IW_{c_1} \text{ or } INW_{c_1, c_2} \geq IW_{c_2} \quad (6)$$

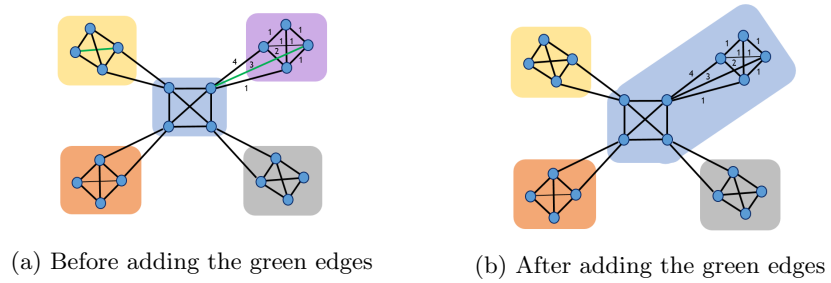
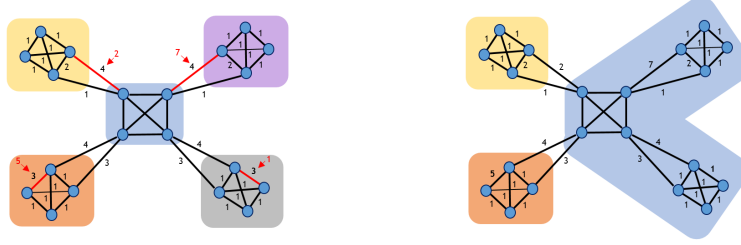


Figure 5: Edge addition scenario example

3.2.5. *EdgeWeightUpdating (edgeWeightUpdate)*:

The last and not least update case can be partitioned into two subcases, illustrated in Figure 6. The first subcase is when the *edgeWeightUpdate* is an inter-community edge with weight greater than the old edge weight. For this scenario *Dyci* verifies whether Equation 6 holds or not. The second subcase rises when *edgeWeightUpdate* is an intra-community edge with weight lower than the old edge weight. The algorithm checks whether this weight loss leads to an adjacent community merging by Equation 5.



(a) Before updating the red edges weights (b) After updating the red edges weights

Figure 6: Edge weight update scenario example

4. Genetic algorithm

Genetic algorithms (GA) can provide very good results if they are well set. In order to evaluate the quality of the obtained communities of Cs_f , a comparison is conducted between *Dyci* and the following GA.

- **Chromosome encoding:** The Locus-based Adjacency Representation [29] (LAR) is used to encode the community detection problem, like in [30], [23]. In the LAR a $|N_f|$ sized array is used, where the couples (gene, allele) express an associative community membership. Indeed, each gene takes its allele value from the set of its node neighbours ensuring feasible solutions. Figure 7 shows an example with the related individual decoding.
- **Fitness function:** Modularity φ of [26] defined by Equation 7 is used for individual evaluation.

$$\varphi = \frac{1}{2M} \sum_{i=1}^v \sum_{j=1}^v \left(e_{n_i, n_j}^w - \frac{W D_{n_i} W D_{n_j}}{2M} \right) \delta(c_{n_i}, c_{n_j}) \quad (7)$$

Where:

- $M = \sum_{i=1}^v \sum_{j=i+1}^v e_{n_i, n_j}^w$
- $\delta(c_{n_i}, c_{n_j}) = 1$, if $c_{n_i} = c_{n_j}$, 0 otherwise.

The modularity expresses whether the detected community structure is well defined or not, corresponding to the density of the detected communities minus the density of these communities for the random case with the same characteristics. As an instance, let suppose that the edge weight set for the graph of Figure 7 is defined as follows:

$$- \{E_t^w\}^{|E_t|} = 1, \text{ thus } M = 13 \text{ and } \varphi = 0.423.$$

- **Population initialization:** A random population of size 100 is generated and sorted in a decreasing fitness function order.
- **Crossover:** Uniform crossover with probability 0.9 is performed, as illustrated in Figure 8a.
- **Mutation:** Random allele flipping with probability 0.1 is performed, as showed in Figure 8b.
- **Parent selection and child insertion:** Random selection from the 20% eliteness individuals. Weakest individuals are excluded from the population.
- **Stopping condition:** Number of generations reaches 50.

As previously mentioned, the LAR is used as encoding because it ensures feasible solution solutions with the induced (gene, allele) associative community membership and because it provides a straightforward decoding step (i.e. linear with respect the size of the individual chromosome). Additionally, the chosen crossover and mutation operators maintain the feasibility solution of the LAR encoding.

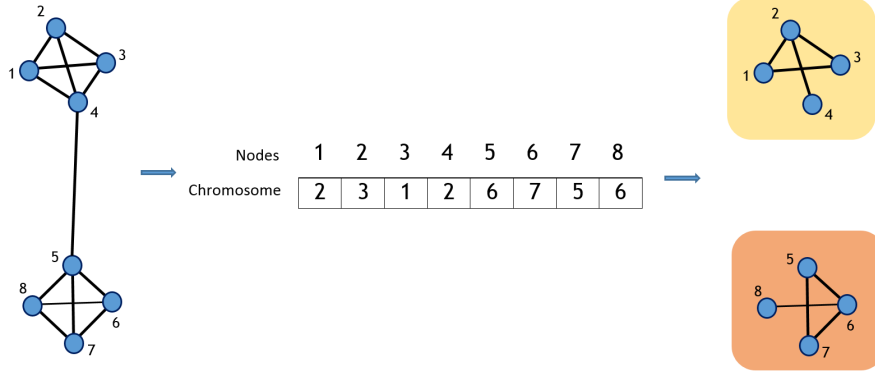


Figure 7: An individual example using LAR encoding

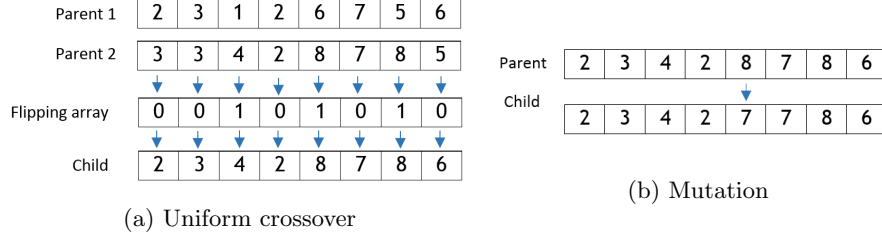


Figure 8: Reproduction operators

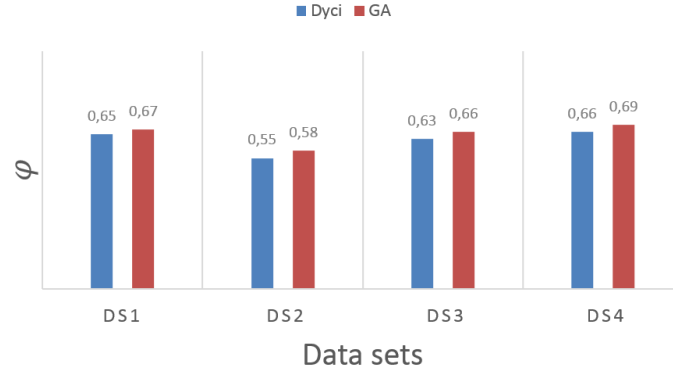
Table 1: The ANR-Info-RSN datasets characteristics

Data sets	t_0	t_f	$ N_s $	$ E_s $	$ N_f $	$ E_f $
DS1	July 17,2014	July 31,2014	10569	14121	801	997
DS2	August 3,2014	August 15,2014	6162	8069	390	451
DS3	August 17,2014	August 31,2014	10189	12263	424	508
DS4	September 3,2014	September 30,2014	8224	10371	412	535

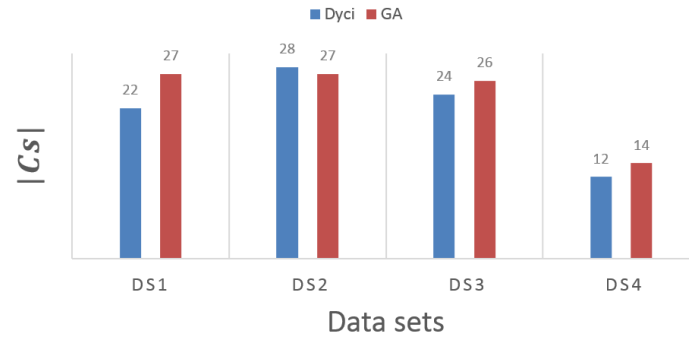
5. Twitter’s network and experimentation

250 This section discusses the obtained results of the conducted comparison between the previous GA and *Dyci* on four datasets from real-world data of the ANR-Info-RSN project. The ANR-Info-RSN project deals with the community detection in Twitter, especially in a huge collected set of tweets from social media. To this end, a graph is used as model, where each Twitter’s user of the collected data is represented by a node and an edge represents a retweet relationship between two Twitter’s users. In this context, the edge weight is equal to the number of times where a retweet is observed between two Twitter’s users. Table 1 presents the datasets characteristics where the unit of snapshot generation is one day. Figure 9 and Figure 10 show, respectively, the obtained results for the datasets at t_f and their averages values for the datasets for the whole Cs_s .

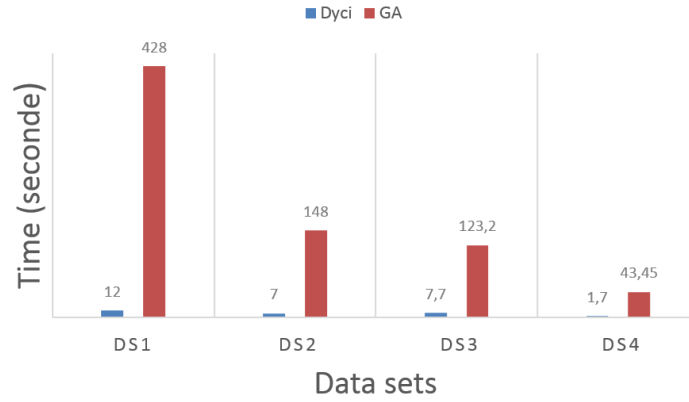
From Figure 9a, we remark that *Dyci* and the GA have almost the same results (GA very slightly better), taking into account the fact that the obtained communities Cs_f of *Dyci* are highly influenced by the f previous choices made during the whole graph sequence. One could say that *Dyci* obtains satisfactory results. Further, from Figure 9c, we remark that *Dyci* is relatively fast compared to the GA, due to the fact that *Dyci* takes advantage from the previous identified community avoiding relaunching the process at each snapshot. From Figure 10, we notice that the averages values are almost the same by comparing to the values of the last snapshot t_f , except for DS3 where *Dyci* takes more time and provides less modularity for the previous snapshots but has relatively a good result for the last snapshots t_f .



(a) Obtained modularity

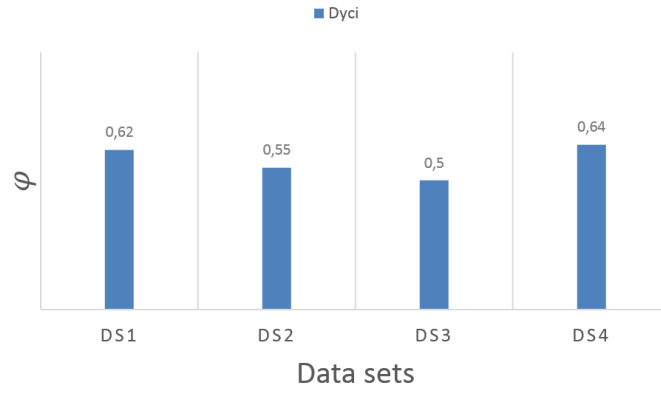


(b) Identified number of communities

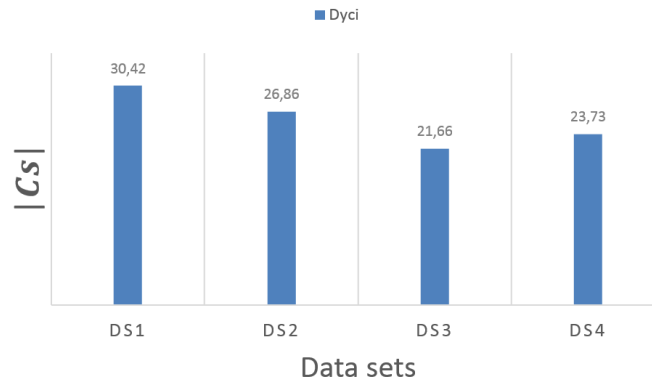


(c) Running time

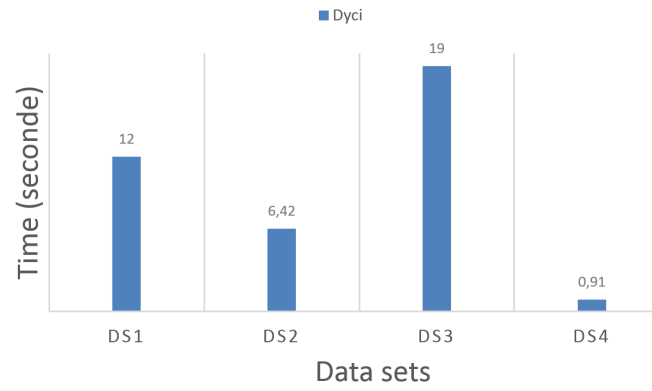
Figure 9: The results for the ANR-Info-RSN data sets at t_f



(a) Obtained modularity



(b) Identified number of communities



(c) Running time

Figure 10: The results for the ANR-Info-RSN data sets for the whole Cs_s

6. Dynamic community visualization with *NLCOMS*

After handling the set of updates U_t and thus, inferring the impacts that cause to the community set leading to $C_{s_{t+1}}$, one has to draw the related graph G_{t+1} . In this work, we use the previous layout L_t of G_t as a starting point to embed the graph G_{t+1} with L_{t+1} . To this end, we use our visualization tool, called *NLCOMS* (Node-Link and COMmunitieS), which relies on node-link diagram and force-directed algorithm [24] for dynamic graph drawing while providing three node positioning possibilities:

- **Free:** the charges of the force-directed algorithm are applied on all the nodes which induces a freely node positioning.
- **Fixed:** The nodes present in the previous layout L_t are kept in the same position in L_{t+1} .
- **Anchored:** This idea was already introduced by [13] in their absolute paradigm. Here, a virtual node is linked to each node which was in the previous layout L_t acting like a boat anchor.

Figure 11 illustrates a sample graph comparison of the previous three node positioning possibilities using *NLCOMS*. From Figure 11 we can remark that the three node positioning possibilities behave differently. If the user’s visual task is to detect a specific node through the different snapshots, it is obvious that the freest way to layout node might lead to misleading interpretation with temporal alias [9] (i.e., in the node-link context, nodes mistaken one for the other due to their positioning in different snapshots). The fixed node position is efficient for relatively small graphs, however it suffers from scalability issue. Additionally, the final node position strongly depends on the initial node position. As an example, in the layout at L_{t+2} , the uniformity of the edge length and the edge crossing criteria are not met. The third possibility, the anchored node position, provides a pleasing graph drawing manner and ensures a good trade-off between user’s mental map preservation and graph drawing aesthetics.

The next step of the proposed approach in this work consists in displaying the successive graph layouts in the limited depiction area. To this end, *NLCOMS* uses the physical time and the axial time in order to take advantage from both of them. Figure 12 shows a sample from the real-world data of ANR Info-RSN database from October 8, 2014 to October 10, 2014. The two last snapshots are depicted allowing the user to scroll down and up to explore the historic of the graph structure. As the graphs are generated from the initial graph G_0 , the user would like to perceive the presence of the initial nodes. For that, the shape is used as a visual variable [11] to fulfill this task, represented with triangles. Additionally, *NLCOMS* highlights nodes’ presence frequency over time via the node inner color saturation. Indeed, the darker a node is, the more frequent it was observed through the previous snapshots. This visual task provides to the user a perception of nodes persisting over successive snapshots and those which are just outliers at a specific snapshot. However, even if they are

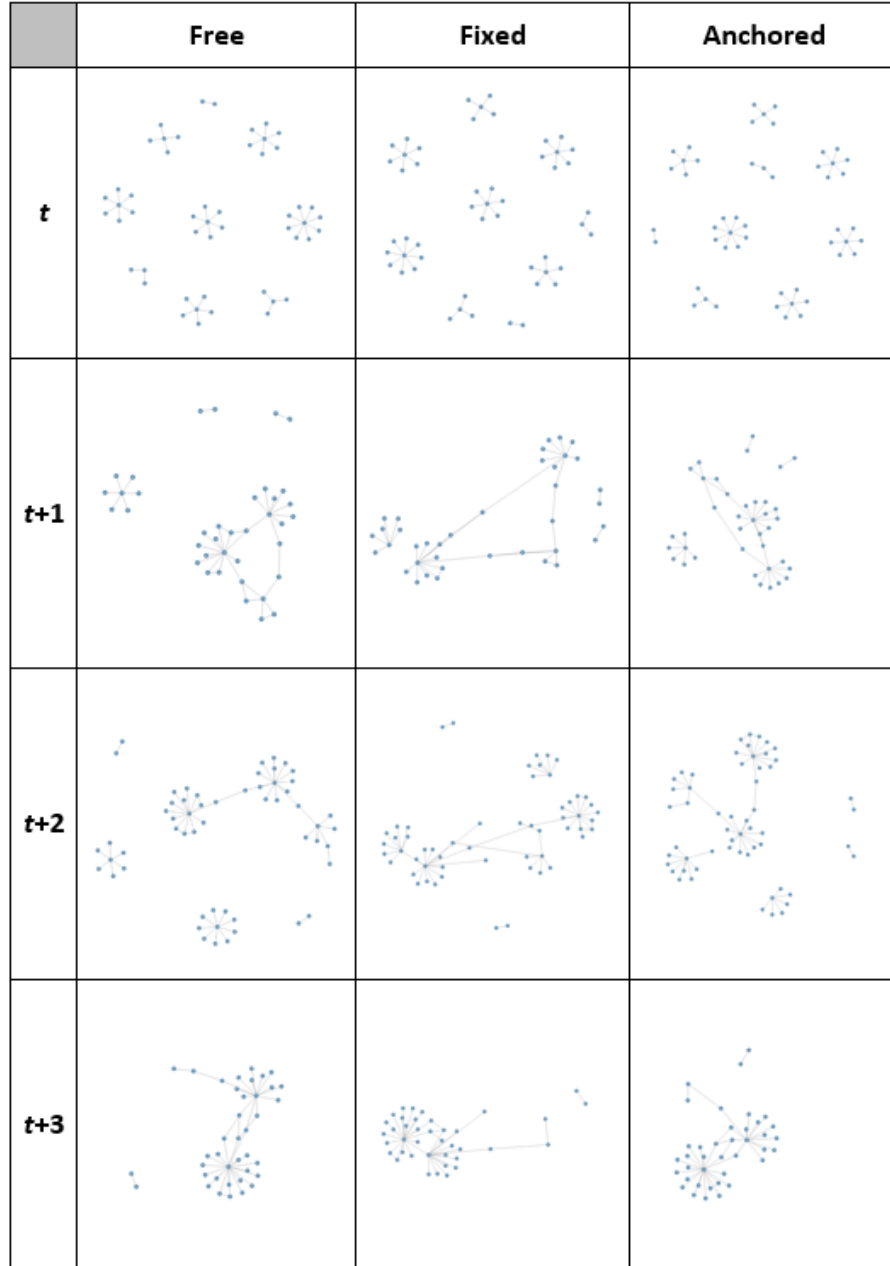


Figure 11: A comparison between node position possibilities drawn using *NL-COMS*

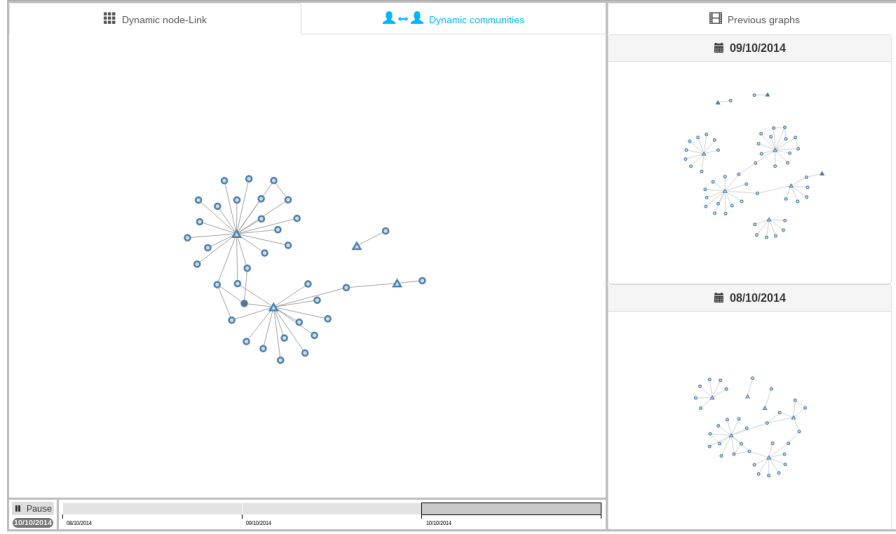


Figure 12: Multiple views of *NLCOMS* for dynamic graph visualization

outliers at a specific instant t , they may contribute to the overall graph structure understanding. Finally, a different color is assigned to the nodes which belong to different communities detected by *Dyci*. This ensures community membership distinction as user visual task.

7. Conclusion

Dynamic social network analysis attracts more interest from the scientific community. Indeed, understanding of the graph structure over time is a very challenging task. In this work, we propose an approach for analysing dynamic social networks, more precisely for Twitter's network. Our approach relies on two complementary steps: (i) an online community identification based on a dynamic community detection algorithm called *Dyci*. The latter checks over time whether a connected component of the weighted graph becomes weak in terms of weight, in order to proceed to a local community re-identification. Thus, community detection from scratch at each snapshot is avoided. (ii) a community visualization is provided by *NLCOMS*, which combines two methods (i.e., physical and axial time) of dynamic network visualization. The efficiency of the proposed approach is assessed and a comparison study is conducted with a genetic algorithm on samples from real-world data of the ARN-Info-RSN project, which deals with community analysis in Twitter. The results show that our proposed approach allows us to efficiently detect and to visualize the underlying communities to the network structure.

As perspectives, we project to extend this work to multi-graphs with several edges types linking two nodes. As an instance, considering simultaneously

retweet edges and "mention" edges. The latter exists when a Twitter's user quotes another Twitter's user in its tweet. In this context, community detection algorithm could consider overlapping communities.

Acknowledgements

This research has been supported by the Agence Nationale de la Recherche (ANR, France) during the Info-RSN Project (ANR-13-SOIN-0008).

References

References

- [1] Abdelsadek, Y., Chelghoum, K., Herrmann, F., Kacem, I., & Otjacques, B. (2015). Community detection algorithm based on weighted maximum triangle packing. In *Proceedings of International Conference on Computer and Industrial Engineering CIE45*.
- [2] Aigner, W., Miksch, S., Schumann, H., & Tominski, C. (2011). *Visualization of Time-Oriented Data*. Human-Computer Interaction Series. Springer. URL: <http://dx.doi.org/10.1007/978-0-85729-079-3>.
- [3] Aktunc, R., Toroslu, I. H., Ozer, M., & Davulcu, H. (2015). A dynamic modularity based community detection algorithm for large-scale networks: Dslm. In J. Pei, F. Silvestri, & J. Tang (Eds.), *ASONAM* (pp. 1177–1183). ACM. URL: <http://dblp.uni-trier.de/db/conf/asunam/asonam2015.html#AktuncTOD15>.
- [4] Alvares, H., Hajibagheri, A., & Sukthankar, G. R. (2014). Community detection in dynamic social networks: A game-theoretic approach. In X. Wu, M. Ester, & G. Xu (Eds.), *ASONAM* (pp. 101–107). IEEE Computer Society. URL: <http://dblp.uni-trier.de/db/conf/asunam/asonam2014.html#AlvaresHS14>.
- [5] Archambault, D., & Purchase, H. C. (2013). The "map" in the mental map: Experimental results in dynamic graph drawing. *Int. J. Hum.-Comput. Stud.*, 71, 1044–1055. URL: <http://dblp.uni-trier.de/db/journals/ijmms/ijmms71.html#ArchambaultP13>.
- [6] Archambault, D., Purchase, H. C., & Pinaud, B. (2010). Difference map readability for dynamic graphs. In U. Brandes, & S. Cornelsen (Eds.), *Graph Drawing* (pp. 50–61). Springer volume 6502 of *Lecture Notes in Computer Science*. URL: <http://dblp.uni-trier.de/db/conf/gd/gd2010.html#ArchambaultPP10>.
- [7] Archambault, D., Purchase, H. C., & Pinaud, B. (2011). Animation, small multiples, and the effect of mental map preservation in dynamic graphs. *IEEE Trans. Vis. Comput. Graph.*, 17, 539–552. URL: <http://dblp.uni-trier.de/db/journals/tvcg/tvcg17.html#ArchambaultPP11>.

- [8] Bansal, S., Bhowmick, S., & Paymal, P. (2010). Fast community detection for dynamic complex networks. In L. da F. Costa, A. Evsukoff, G. Mangioni, & R. Menezes (Eds.), *CompleNet* (pp. 196–207). Springer volume 116 of *Communications in Computer and Information Science*. URL: <http://dblp.uni-trier.de/db/conf/complenet/complenet2010.html#BansalBP10>.
- [9] Beck, F., Burch, M., & Diehl, S. (2009). Towards an aesthetic dimensions framework for dynamic graph visualisations. In *Information Visualisation, 2009 13th International Conference* (pp. 592–597). IEEE.
- [10] Beck, F., Burch, M., Diehl, S., & Weiskopf, D. (2014). The state of the art in visualizing dynamic graphs. *EuroVis STAR*, .
- [11] Bertin, J. (1967). *Sémiologie graphique : les diagrammes, les réseaux, les cartes*. Paris: Mouton.
- [12] Blondel, V., Guillaume, J., Lambiotte, R., & Mech, E. (2008). Fast unfolding of communities in large networks. *J. Stat. Mech*, (p. P10008).
- [13] Brandes, U., & Wagner, D. (1997). A bayesian paradigm for dynamic graph layout. In G. D. Battista (Ed.), *Graph Drawing* (pp. 236–247). Springer volume 1353 of *Lecture Notes in Computer Science*. URL: <http://dblp.uni-trier.de/db/conf/gd/gd1997.html#BrandesW97>.
- [14] Branke, J. (2001). Drawing graphs. chapter Dynamic Graph Drawing. (pp. 228–246). London, UK, UK: Springer-Verlag. URL: <http://dl.acm.org/citation.cfm?id=376944.376953>.
- [15] Diehl, S., & Görg, C. (2002). Graph drawing: 10th international symposium, gd 2002 irvine, ca, usa, august 26–28, 2002 revised papers. chapter Graphs, They Are Changing. (pp. 23–31). Berlin, Heidelberg: Springer Berlin Heidelberg. URL: http://dx.doi.org/10.1007/3-540-36151-0_3. doi:10.1007/3-540-36151-0_3.
- [16] Erten, C., Harding, P. J., Kobourov, S. G., Wampler, K., & Yee, G. V. (2003). Graphael: Graph animations with evolving layouts. In G. Liotta (Ed.), *Graph Drawing* (pp. 98–110). Springer volume 2912 of *Lecture Notes in Computer Science*. URL: <http://dblp.uni-trier.de/db/conf/gd/gd2003.html#ErtenHKWY03>.
- [17] Fortunato, S. (2010). Community detection in graphs. *Physics Reports*, 486, 75 – 174. URL: <http://www.sciencedirect.com/science/article/pii/S0370157309002841>. doi:<http://dx.doi.org/10.1016/j.physrep.2009.11.002>.
- [18] Freeman, L. (2004). *The Development of Social Network Analysis: A Study in the Sociology of Science*. BookSurge Publishing.

- [19] Frishman, Y., & Tal, A. (2007). Online dynamic graph drawing. In K. Museth, T. Mller, & A. Ynnerman (Eds.), *EuroVis* (pp. 75–82). Eurographics Association. URL: <http://dblp.uni-trier.de/db/conf/vissym/eurovis2007.html#FrishmanT07>.
- [20] Greene, D., Doyle, D., & Cunningham, P. (2010). Tracking the evolution of communities in dynamic social networks. In *Proceedings of the 2010 International Conference on Advances in Social Networks Analysis and Mining ASONAM '10* (pp. 176–183). Washington, DC, USA: IEEE Computer Society. URL: <http://dx.doi.org/10.1109/ASONAM.2010.17>. doi:10.1109/ASONAM.2010.17.
- [21] Harary, F., & Gupta, G. (1997). Dynamic graph models. *Math. Comput. Model.*, 25, 79–87. URL: [http://dx.doi.org/10.1016/S0895-7177\(97\)00050-2](http://dx.doi.org/10.1016/S0895-7177(97)00050-2). doi:10.1016/S0895-7177(97)00050-2.
- [22] Hopcroft, J., Khan, O., Kulis, B., & Selman, B. (2004). Tracking evolving communities in large linked networks. *PNAS*, .
- [23] Jin, D., He, D., Liu, D., & Baquero, C. (2010). Genetic algorithm with local search for community mining in complex networks. In *ICTAI (1)* (pp. 105–112). IEEE Computer Society. URL: <http://dblp.uni-trier.de/db/conf/ictai/ictai2010-1.html#JinHLB10>.
- [24] Kobourov, S. G. (2013). Force-directed drawing algorithms. In R. Tamassia (Ed.), *Handbook of Graph Drawing and Visualization* (pp. 383–408). Chapman and Hall/CRC. URL: <http://dblp.uni-trier.de/db/reference/crc/gd2013.html#Kobourov13>.
- [25] Leskovec, J., Rajaraman, A., & Ullman, J. D. (2014). *Mining of Massive Datasets, 2nd Ed.*. Cambridge University Press. URL: <http://www.mmms.org/>.
- [26] Newman, M. (2006). Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103, 8577–8582.
- [27] Newman, M. E. J., & Girvan, M. (2004). Finding and evaluating community structure in networks. *Physical Review, E* 69.
- [28] Nguyen, N. P., Dinh, T. N., Xuan, Y., & Thai, M. T. (2011). Adaptive algorithms for detecting community structure in dynamic social networks. In *INFOCOM* (pp. 2282–2290). IEEE. URL: <http://dblp.uni-trier.de/db/conf/infocom/infocom2011.html#NguyenDXT11>.
- [29] Park, Y., & Song, M. (1998). A genetic algorithm for clustering problems. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, & R. Riolo (Eds.), *Genetic Programming 1998: Proceedings of the Third Annual Conference* (pp. 568–575). University of Wisconsin, Madison, Wisconsin, USA: Morgan Kaufmann.

- [30] Pizzuti, C. (2008). Ga-net: A genetic algorithm for community detection in social networks. In *PPSN* (pp. 1081–1090). Springer volume 5199 of *Lecture Notes in Computer Science*.
- [31] Pupyrev, S., & Tikhonov, A. (2010). Analyzing conversations with dynamic graph visualization. In *10th International Conference on Intelligent Systems Design and Applications, ISDA 2010, November 29 - December 1, 2010, Cairo, Egypt* (pp. 748–753). URL: <http://dx.doi.org/10.1109/ISDA.2010.5687175>. doi:10.1109/ISDA.2010.5687175.
- [32] Rufiange, S., & McGuffin, M. J. (2013). Diffani: Visualizing dynamic graphs with a hybrid of difference maps and animation. *IEEE Trans. Vis. Comput. Graph.*, 19, 2556–2565. URL: <http://dblp.uni-trier.de/db/journals/tvcg/tvcg19.html#RufiangeM13>.
- [33] Takaffoli, M., Sangi, F., Fagnan, J., & Zane, O. R. (2011). Community evolution mining in dynamic social networks. *Procedia - Social and Behavioral Sciences*, 22, 49 – 58. URL: <http://www.sciencedirect.com/science/article/pii/S1877042811013784>. doi:<http://dx.doi.org/10.1016/j.sbspro.2011.07.055>. Dynamics of Social Networks7th Conference on Applications of Social Network Analysis - {ASNA} 2010.
- [34] Tantipathananandh, C., Berger-Wolf, T., & Kempe, D. (2007). A framework for community identification in dynamic social networks. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 717–726). New York, NY, USA: ACM. URL: <http://portal.acm.org/citation.cfm?doid=1281192.1281269>. doi:<http://doi.acm.org/10.1145/1281192.1281269>.
- [35] Tantipathananandh, C., & Berger-Wolf, T. Y. (2009). Constant-factor approximation algorithms for identifying dynamic communities. In J. F. E. IV, F. Fogelman-Souli, P. A. Flach, & M. Zaki (Eds.), *KDD* (pp. 827–836). ACM. URL: <http://dblp.uni-trier.de/db/conf/kdd/kdd2009.html#TantipathananandhB09>.
- [36] Tufte, E. R. (1990). *Envisioning Information*. Cheshire: Graphics Press.
- [37] Wasserman, S., & Faust, K. (1994). *Social network analysis: Methods and applications* volume 506. Cambridge University Press.
- [38] Xie, J., Chen, M., & Szymanski, B. K. (2013). Labelrank: Incremental community detection in dynamic networks via label propagation. *CoRR*, abs/1305.2006. URL: <http://dblp.uni-trier.de/db/journals/corr/corr1305.html#abs-1305-2006>.